1 Introduction

1.1 Trois points de vue sur la notation objet

- Abstraction (type abstrait) - point de vue structure
  - Cache les structures (ex : pile) par les manipulations légales (ex : empiler, dépiler)
  - On parle *d'encapsulation* de l'objet
- Réseaux sémantiques (Intelligence artificielle) - point de vue concept
  - Sous classe SC *est une sorte de classe* C
  - Objet O *est un (est une instance)* de C
- Langage d'acteurs (Intelligence artificielle) - point de vue dynamique
  - l'objet O reçoit un message et y réagit
  - Système multi-agents

1.2 Trois grandes classes d'applications

- La conception = représenter le monde réel pour l'informatiser
  - Représente les entités (classe) et les liens (héritage) naturellement
  - Visualisation facile (graphe des classes) synthèse et modifications
- La programmation = le codage
  - La programmation jaillit de la modélisation
  - Les fonctions sont relatives aux données
  - Un même nom de fonction pour même opération, même si codage différent (langage abstrait)
- Les bases de données = la persistance
  - Pourquoi utiliser des tables ?
  - Objets persistants

1.3 Principes

- Principe 1 : on raccroche à la structure de données (champs), les codes qui peuvent les manipuler (méthode) -> classe
- Principe 2 : on hiérarchise les classes -> héritage
- Principe 3 : on assure la sécurité des accès aux données et méthodes sous forme d'autorisation ; on gère les erreurs

2 Propriétés des accès

Qui peut accéder aux champs, méthodes et classes ?

2.1 Le constructeur

- Appliqué à une classe retourne un objet de la classe
  - Créer un objet à partir d'une classe
  - Initialiser l'objet par les valeurs adéquates (initialiser après le new les champs)
- Programmer des constructeurs paramétrés
  - (LOLO) : nomClasse <= new(val1,..,valn)
  - (en JAVA) : définir des méthodes du nom de la classe
    - Quand aucun constructeur n'est défini -> constructeur par défaut
    - Le constructeur par défaut est désactivé dès qu'il existe un constructeur programmé
    - On peut reconstruire à la main un constructeur sans paramètre
- L'utilisation de *super*
  - super() appel le constructeur de la sur-classe
  - super.méthode(…) appel de méthode de la classe supérieur

2.2 Accès privé/public

L'accès à un champs peut être :

- **privé** : seules les méthodes de la classe C peuvent utiliser le champs
- **public** : tout code (ayant accès à la classe C) peut l'utiliser
- **protégé** : seules les méthodes de la classe C et de ses sous classes peuvent utiliser le champs

Règles :

- un champs est généralement privé
- –> accès par des méthodes adéquates
- un champs gérant des données d'implémentation est privé

L'accès aux méthodes m dd'une classe C peut être :

- **privé** : seules les méthodes de la classe C peuvent utiliser m
- **public** : tout code (ayant accès à la classe C) peut utiliser m
- **protégé** : seules les méthodes de la classe C et de ses sous classes peuvent utiliser m

Remarques :

- *private, public, protected* (en JAVA) sont appelés des modificateurs (d'accès)
- Accesseurs (conventions)
  - *getVar* : accès lecture à un champs *Var*
  - *setVar* : accès écriture à un champs *Var*

On parle de portée d'une déclaration

2.3 Les paquetages

Un paquetage, en JAVA, n'existe pas en Smalltalk ou C++

- Idées
  - pour de grandes applications
  - regroupe des classes dans un paquetage (*package* en JAVA)
  - Nom de la classe, référé à un paquetage (permet 2 classes de même nom)
  - Les paquetages organisent logiquement les classes dans une hiérarchie ; généralement, l'organisation physique est la même
- Principes du package (logique) et de son positionnement système (physique)
  - Un package est associé à un répertoire :le nom du pakage est celui du répertoire
  - Une hiérarchie de packages correspond à une hiérarchie de répertoires
  - Dans un package, il y a des classes, et des sous packages (= sous répertoires)
  - Le premier niveau de hiérarchie correspond au nom de l'application ou du propriétaire (ex : loiseau, java)
- Principes de nommage, d'appartenance et de recherche
  - un mécanisme de nommage : *nomPackage.nomSousPackage.NomClasse*

  - appartenance :
    * Par défaut : une classe dans un répertoire appartient au package correspondant
    * Par déclaration : package nomPackage.nomSousPackage (1ere instr)
  - recherche : variable *CLASSPATH* (Windows)

- Accès aux classes d'un package
  - les classes (object, String...) du package java.lang sont utilisables dans toute classe
  - pour importer une classe C ou un packetage P dans un programme : import C et import P.* en tête de fichier
  - Si pas de modificateur de champs et méthode : toutes les classes du package y ont accès
  - Attention : Modificateur d'accès d'une classe C, *public* : la classe est accessible en dehors du package où elle est définie pas de modificateur : seules les classes du même paquetage ont accès à C

3 Les classes pour elles-même

3.1 Retour à l'héritage

- l'autoréférencement
  - pour appliquer une méthode à l'objet
  - pour lever l'ambiguïté entre nom de champs et variable
- L'option *final* (sécurité)
  - une méthode *final* ne peut pas être redéfinie dans une sous classe
  - une classe définie *final* ne peut pas être héritée
  - une variable *final* est une constante

3.2 Classe abstraite

Définition 1 *Une classe abstraite est une classe pour laquelle il ne peut exister d'instances.*

-